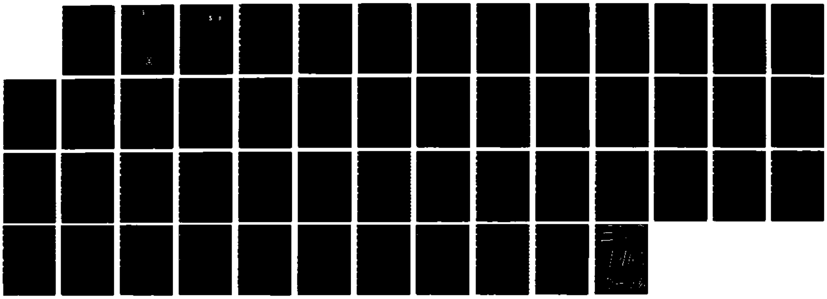
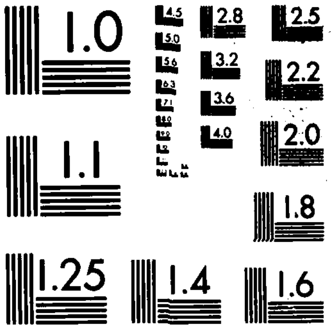
AD-A166 046 PARALLEL SOLUTION OF LINEAR SYSTEMS WITH STRIPED SPARSE MATRICES PART 1 V (U) PITTSBURGH UNIV PA INST FOR COMPUTATIONAL MATHEMATICS AND APP R MELHEM JAN 86 1/1

UNCLASSIFIED ICMA-86-91 N00014-85-K-0339 F/G 12/1 NL




1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0
4.5
5.0
5.6
6.3
7.1

(14)

INSTITUTE FOR COMPUTATIONAL MATHEMATICS AND APPLICATIONS

Technical Report ICMA-86-91 January, 1986

PARALLEL SOLUTION OF LINEAR SYSTEMS WITH STRIPED SPARSE MATRICES *

PART 1: VLSI networks for striped matrices

by

Rami Melhem **

Department of Mathematics and Statistics
University of Pittsburgh

4

Technical Report ICMA-86-91 January, 1986

PARALLEL SOLUTION OF LINEAR SYSTEMS WITH STRIPED SPARSE MATRICES *

PART 1: VLSI networks for striped matrices

by

Rami Melhem **

Institute for Computational Mathematics and Applications

Department of Mathematics and Statistics
University of Pittsburgh

* This work is, in part, supported under ONR contract N00014-85-K-0339

** On leave from the Department of Computer Science, Purdue University, West Lafayette, IN 47907

PARALLEL SOLUTION OF LINEAR SYSTEMS WITH STRIPED SPARSE MATRICES

PART 1: VLSI networks for striped matrices

Rami Melhem

ABSTRACT

The multiplication of a matrix by a vector and the solution of triangular linear systems are the most demanding operations in the majority of iterative techniques for the solution of linear systems. Data-driven VLSI networks that perform these two operations, efficiently, for sparse matrices are introduced. In order to avoid computations that involve zero operands, the non-zero elements in a sparse matrix are organized in the form of non intersecting stripes, and only the elements within the stripe structure of the matrix are manipulated. Detailed analysis of the networks proves that both operations may be completed in n global cycles with minimal communication overhead, where n is the order of the linear system. The number of cells in each network as well as the communication overhead are determined by the stripe structure of the matrix. A companion paper [12] examines this structure for the class of sparse matrices generated in Finite Element Analysis.

*) This work is, in part, supported under ONR contract N00014-85-K-0339.

**) On leave from the Dept. of Computer Science, Purdue Univ., West lafayette, IN 47907.

1. Introduction

Iterative solvers for large sparse linear systems of equations are, once again, becoming more and more competitive with direct solvers [6,7]. In this paper, we consider the two basic operations that constitute the bulk of the work in most iterative methods. Namely, the multiplication of a matrix by a vector, and the solution of triangular linear systems. The computations involved in these two operations are quite regular and thus, naturally amenable to efficient implementations on regular VLSI networks; that is systolic [9] and data-driven (sometimes called self-timed) [10] arrays.

However, large systems that appear in practice are usually sparse, and hence, seem to be inefficient for solution on regular VLSI networks. More specifically, if ζ% of the elements in the coefficient matrix are zeroes, then ζ% of the resources in the network are wasted in trivial operations that involve zero operands, and the corresponding data communications.

In order to avoid this waste and to retain the advantage of having fast specialized cells and efficient local communications, it is suggested in [13] to consider regular data-driven networks that are designed for operations on dense matrices, and then to add to each cell in the networks the capability of recognizing and skipping trivial operations. A detailed study of the performance of the resulting networks shows large speed ups for highly sparsed matrices.

In this paper, we present a different approach for using regular data-driven networks in sparse matrix manipulation. It is also based on performing only non-trivial operations, but is primarily aimed at reducing the number of computational cells in the network, rather than increasing its speed.

In order to be more specific, consider, for example, the multiplication of an n×n banded matrix A by a vector. This operation may be performed in 2n+b cycles on a data-driven (or systolic) network [9] that uses b cells, where b is the band width of A. If A is sparse, then the approach of [13] uses the same number of cells b, but takes advantage of the sparsity of A to reduce the multiplication time considerably. On the other hand, the approach presented here takes advantage of the sparsity of A to reduce the number of cells to a number π, which is, usually, much smaller than b. The multiplication, in this case, is completed in approximately n cycles.

The reduction of the number of computational cells is based on the assumption that the non-zero elements of the matrix are located in a few stripes which are almost parallel to the diagonal of the matrix. A specific cell is then assigned to perform the operations associated with the elements of a particular stripe. Clearly, the crucial issue is to choose a stripe structure that minimizes, or even eliminates, data conflict.

The network that we introduce here is especially suitable for the type of matrices that result in finite element analysis. More specifically, for this type of matrices, the band-width depends on the order of the matrix (usually, $b=O(\sqrt{n})$), while the number of stripes π is bounded by a small number which depends on the maximum number of elements that may share a particular node. In other words, unlike in [13], the size of the network is independent of the size of the problem.

At this point, we should mention that other different approaches have been suggested for the parallel solution of sparse linear systems. These include the application of content-addressable VLSI networks [18], and data flow architectures [15,16] to the minimization of conflicts in data access, the use of networks with interconnections that reflect the underlying graph structure of the matrix [1,2], and the use of multiprocessors with general interconnections [3,4].

We start in Section 2 by defining the stripe structure of a general sparse matrix. In Section 3, we describe a VLSI network that utilizes the stripe structure in the parallel multiplication of a matrix by a vector. Then, we introduce, in Section 4, the property of non-overlapping stripes and we show that, if this property is satisfied in the input matrix, then the multiplication does terminate in n global cycles, where a global cycle includes some communication activities, and a multiply/add operation. In Section 5, we estimate the communication overhead in each global cycle, and finally, in Section 6, we modify the matrix/vector multiplication network and obtain a network for the solution of triangular linear systems.

2. STRIPE STRUCTURES OF SPARSE MATRICES

We define a stripe structure of a sparse matrix to be a set of stripes that are almost parallel to the diagonal of the matrix, and that contains all its non-zero elements. More specifically, given an $n \times n$ matrix A, with lower and upper bandwidthes b_1 and b_2, respectively, we augment the set $T = \{(i,j);\ 1 \leq i,j \leq n\}$ of positions of A with the two triangles $T_1 = \{(i,j);\ i=1,\ldots,b_1, j=i-b_1,\ldots,0\}$ and $T_2 = \{(i,j);\ i=n-b_2+1,\ldots,n,\ j=n+1,\ldots,i+b_2\}$, and assume that $a_{i,j}=0$ for $(i,j) \in T_1 \cup T_2$. This expands the set of allowable positions of A to include the band $\{(i,j);\ 1 \leq i \leq n,\ i-b_1 \leq j \leq i+b_2\}$. Now we may define the following:

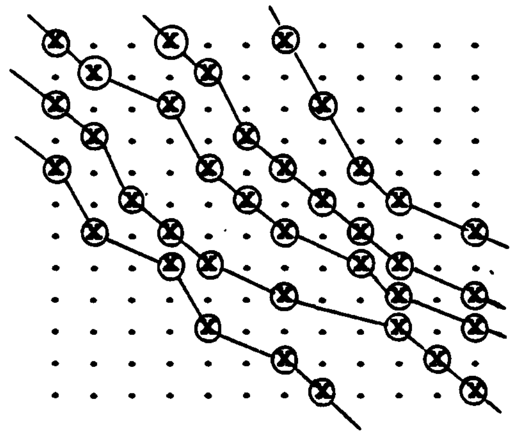
Definition 1: Let $I_n=\{1,\ldots,n\}$. A stripe S of the matrix A is a set of positions $S = \{(i,\sigma(i))\ ;\ i \in I \subseteq I_n\}$, where σ is an increasing function; that is, if $i < j$ and $(i,\sigma(i)),(j,\sigma(j)) \in S$, then $\sigma(i) < \sigma(j)$. If S contains one entry for each row of A; that is $S = \{(i,\sigma(i))\ ;\ i \in I_n\}$, then S is called a full stripe.

Definition 2: Two stripes $S_1 = \{(i,\sigma_1(i))\}$ and $S_2 = \{(i,\sigma_2(i))\}$ are ordered by the relation $S_1 < S_2$ (S_1 is less than S_2) if for any i in the domain of σ_1, and j in the domain of σ_2,

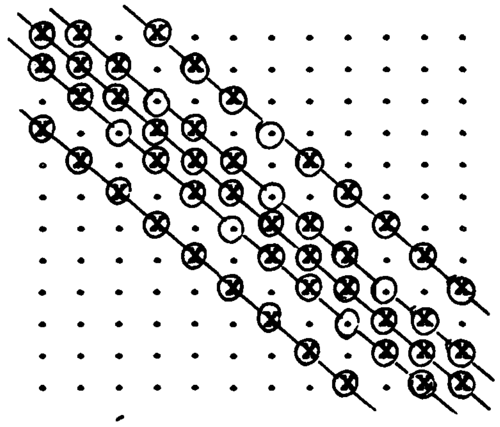
$$i \leq j \quad \text{implies that} \quad \sigma_1(i) < \sigma_2(j).$$

Note that if S_1 and S_2 are full stripes, then $S_1 < S_2$ if $\sigma_1(i) < \sigma_2(i)$ for $i=1,\ldots,n$.

<u>Definition 3</u>: A stripe structure of a matrix A is a sequence of π stripes $S_1 < \ldots < S_\pi$, such that $a_{i,j}=0$ if $(i,j) \notin S_1 \cup \ldots \cup S_\pi$ (see Fig 1(a), where '.' and 'x' indicate a zero and a non-zero element, respectively, and each element included in a stripe is enclosed in a circle).



(a) with 5 stripes



(b) with 5 full, parallel, stripes

Fig 1 - Examples of stripe structures

A special class of stripe structures is the class of structures with parallel stripes, in the sense that each stripe S_k has the form $\{(i,i+s_k)\ ;\ i \in I \subseteq I_n\}$, for some constant s_k. Matrices that can be covered by parallel stripes occur frequently in practice, especially in the solution of partial differential equations on rectangular grids. For example, if the nodes of the grid are numbered regularly, and a five point star approximation is used to discretize the differential equation, then the resulting coefficient matrix may be covered by five parallel full stripes (see Fig 1(b)). Similarly, if finite element analysis is used with 3, 4, 6 or 9 node Lagrangian elements, then the resulting stiffness matrix may be covered by 7, 9, 19 or 25 parallel full stripes, respectively. Note that in order to obtain full stripes in the examples of Fig 1(b), we include in each stripe few positions (i,j) for which $a_{i,j}=0$. Note also that if the finite element grid is not rectangular, which is usually the case, then the resulting matrix may not be covered by parallel stripes.

Matrices with parallel full stripes may be efficiently stored diagonal by diagonal. This diagonal storage scheme may be easily extended to matrices with general stripes. More specifically, given a matrix A with π stripes, we may store the elements of the k^{th} stripe in the k^{th} column of an $n\times\pi$ rectangular array E_A, such that, for $i=1,\ldots,n$ and $k=1,\ldots,\pi$,

$$E_A(i,k) = \begin{cases} a_{i,\sigma_k(i)} & \text{if } (i,\sigma_k(i))\in S_k \\ 0 & \text{otherwise} \end{cases} \qquad (1.a)$$

In addition to E_A, another $n\times\pi$ integer array P_A is needed in order to store the values of $\sigma_k(i)$. In other words,

$$P_A(i,k) = \begin{cases} \sigma_k(i) & \text{if } (i,\sigma_k(i))\in S_k \\ -b_1 & \text{otherwise} \end{cases} \qquad (1.b)$$

Note that $-b_1$ is not a valid column number in $T \cup T_1 \cup T_2$. Note also that a more efficient scheme for storing the stripes of A may be obtained if we compact every column k of E_A and P_A by storing only the entries corresponding to the elements of S_k, and then keep the compacted columns of E_A and P_A in two linear arrays. Of course, an additional array is needed in this case in order to keep track of the row number of each element in E_A.

Clearly, many stripe structures may be constructed for a given sparse matrix. Among the different structures for any banded matrix is the structure obtained by considering b parallel stripes, where $b = b_1+b_2+1$ is the band-width. However, in order to take full advantage of the above stripe storage scheme, we should determine the stripe structure of the matrix that minimizes the number of stripes. An algorithm that constructs the optimal stripe structure for any specific sparse matrix is given in the Appendix.

The efficiency of the stripe storage scheme for any $n \times n$ matrix A striped with π stripes is given by the ratio $\frac{r_A}{n\pi}$, where r_A is the number of non-zero elements in A. Although, this ratio may be low for general sparse matrices, it is shown in [12] that the stripe scheme is very efficient for the type of matrices resulting from the discretization of partial differential equations. Moreover, the stripe scheme has an important advantage over other sparse schemes [8], namely, it is a regular scheme that may be explored efficiently in parallel processing.

3. MULTIPLICATION OF A STRIPED MATRIX BY A VECTOR

3.1. A VLSI data driven network

The systolic network given in [9] for the multiplication of a banded matrix A with a vector x uses b cells and completes the computation of the product vector y=Ax in 2n+b cycles, where n and b are the order and band-width, respectively, of A. In this section, we modify this network such that if A has π stripes, $\pi \ll b$, then the vector y may be computed using a network of only π computational cells. In order to be consistent with our future notation, we denote the stripes of A by S_k, $k=-\pi_1,\ldots,\pi_2$, where $\pi_1+\pi_2+1=\pi$.

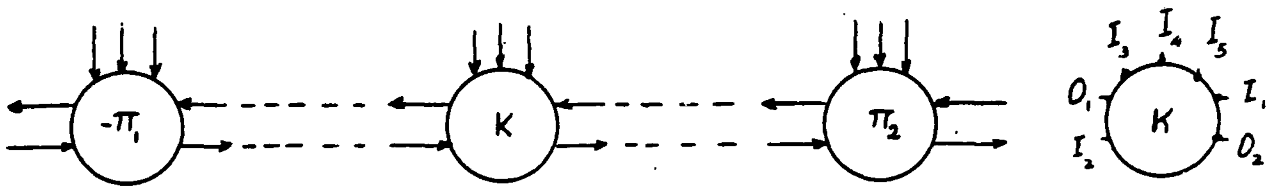


Fig 2 - A network (MAT/VEC) for the multiplication of a striped matrix by a vector

In Fig 2, we show the modified network that we call from now on MAT/VEC. Each cell in MAT/VEC has five input ports, namely I_r, $r=1,..,5$, and two output ports, namely O_1 and O_2. The elements of the vector x are fed to the network from port I_1 of the first cell and the elements of the result-vector y, initialized to zero, are fed from the port

I_2 of the last cell. Successive non-zero elements in a particular stripe S_k are supplied on port I_3 of cell k in increasing row order. Along with each element $a_{i,\sigma_k(i)} \in S_k$ supplied on I_3 of cell k, the values of i and $\sigma_k(i)$ are supplied on the input ports I_5 and I_4, respectively. Note that the row index supplied on I_5 may be eliminated if the stripes are full or if the elements of column k of E_A, defined by (1), rather than the elements of S_k, are supplied to cell k. In this case an internal counter may be used to keep track of the value of i.

Each communication link $\ell_{q,k}$ directed from cell q to cell k in MAT/VEC is regarded as a queue. Only cell q may write on this queue and only cell k may read (and delete) its front element. For simplicity, we will assume, for now, that the queues have unlimited capacity. Then, we will derive in Section 5 the actual size of the queues needed for proper operation. Note that in practice, any communication link is just a connector, and hence, any queues associated with the link should be physically located in its source or destination cells (or distributed among the two).

In order to provide the flexibility needed to deal with sparse structures, we assume that each computational cell contains two counters CX and CY to keep track of the indices of the data received on I_2 and I_1, respectively. We also assume that the network is data driven, that is the cycle of each computational cell is controlled by the availability of the input data. Finally, in order to study the effect of internal data conflicts on the operation of the network, we assume that external data, that is data on ports I_3, I_4 and I_5, as well as port I_1 of cell π_2 and port I_2 of cell $-\pi_1$, are always available when needed. With this, the operation of each cell k may be described by the following cycle which is executed repeatedly by the cell: ($[I_r]$ denotes the content of I_r, and $O_r \leftarrow Rx$ means that the value stored in the internal register Rx is written on port O_r).

```
CYCLE 1:        /* Initially, CX=CY=0 */
    1) Ra = [$I_3$] ; Rj = [$I_4$] ; Ri = [$I_5$]
    2) Do steps 2.1 and 2.2 in parallel
       2.1) wait until data is available on $I_1$
            Rx = [$I_1$] ; CX = CX + 1
            If CX < Rj Then { $O_1 \leftarrow$ Rx ; Go To 2.1 }
                       Else JOIN step 2.2.
       2.2) wait until data is available on $I_2$
            Ry = [$I_2$] ; CY = CY + 1
            If CY < Ri Then { $O_2 \leftarrow$ Ry ; Go To 2.2 }
                       Else JOIN step 2.1.
    3) Ry = Ry + Ra * Rx
    4) $O_1 \leftarrow$ Rx ; $O_2 \leftarrow$ Ry.
```

More descriptively, after a cell k receives $a_{i,\sigma_k(i)}$, (step 1) it continues to transmit the components of x from I_1 to O_1 (step 2.1), and the components of y from I_2 to O_2 (step 2.2), until it finds $x_{\sigma_k(i)}$, and y_i. At this time, the inner product is computed (step 3), and the results are written out (step 4). The JOIN statements in 2.1 and 2.2 indicate that step 3 should not start before both steps 2.1 and 2.2 are completed. Note that the parallel execution of steps 2.1 and 2.2 guarantees that if any of the x or y data streams are blocked, the other stream may continue flowing. This parallel execution may be simulated by a busy wait loop that polls ports I_1 and I_2 for data. More specifically, we may replace step 2 in CYCLE 1 by:

```
2) While $(CX < Rj)$ or $(CY < Ri)$ Do
   2.1) If $(CX < Rj)$ and (data is available on $I_1$) Then
           { $Rx = [I_1]$ ; $CX = CX + 1$ ;
             If $(CX < Rj)$ Then $O_1 \leftarrow Rx$ }
   2.2) If $(CY < Ri)$ and (data is available on $I_2$) Then
           { $Ry = [I_2]$ ; $CY = CY + 1$ ;
             If $(CY < Ri)$ Then $O_2 \leftarrow Ry$ }
```

Next, we show that the network described above does compute the elements y_i, $i=1,\ldots,n$, of the product vector $y=Ax$ correctly if the matrix A has non-intersecting stripes.

3.2. Proof of correctness

The following properties of the input will be used:

P1) If $a_{u,\sigma_k(u)}$ and $a_{v,\sigma_k(v)}$, are the inputs to port I_3 of cell k at two consecutive cycles, t-1 and t, respectively, then $v > u$, P2) From P1 and Definition 1, we have $\sigma_k(v) > \sigma_k(u)$.

P3) The input matrix A is striped according to Definition 3; that is, $S_k < S_q$ if $k<q$.

Let us first assume that the network will not reach a dead state, that is every cycle t of any cell k will terminate. From the operation of each cell (CYCLE 1), and P1, it is clear that every element y_i that is read by cell k (from I_2) during cycle t satisfies $u<i\leq v$. If $i=v$, then the term $[a_{i,\sigma_k(i)}\, x_{\sigma_k(i)}]$ is accumulated in y_i before y_i is written on O_2. On the other hand, if $u<i<v$, then y_i is copied unmodified to O_2. But in this case, P1 guarantees that $(i,\sigma_k(i)) \notin S_k$, because otherwise $a_{i,\sigma_k(i)}$ should have been supplied to I_3 after $a_{u,\sigma_k(u)}$ and before $a_{v,\sigma_k(v)}$. Given that any element of A that does not belong to some stripe is equal to zero, we conclude that y_i accumulates all the non zero terms of $\sum_{j=1}^{n} a_{i,j}\, x_j$ during its flow from cell $-\pi_1$ to cell π_2.

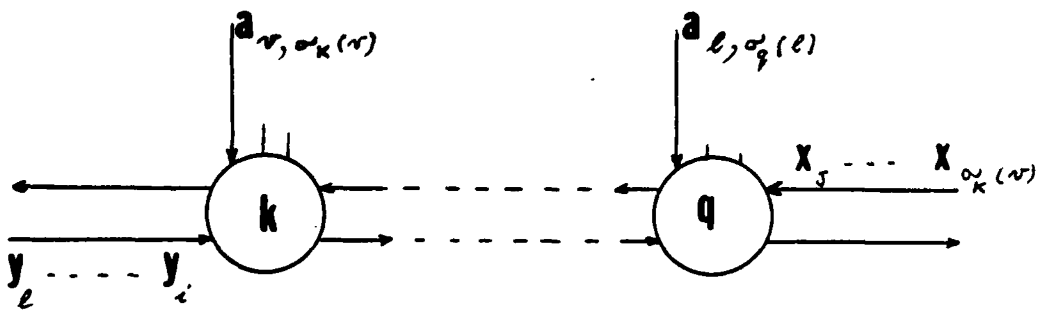


Fig 3 - A deadlock configuration.

In order to complete the proof, we need to show that the network does not reach a deadlock state, where a cell k is blocking the y data stream and another cell q, q>k, is blocking the x stream (see Fig 3). More specifically, a state in which

1) cell q is waiting for some y_ℓ that is locked behind cell k, and

2) cell k is waiting for some $x_{\sigma_k(v)}$ that is locked behind cell q.

Assume that this deadlock state is reached, and that the data appearing on ports I_1 and I_2 of cells q and k, respectively, are x_j and y_i. Hence,

$$\ell \geq i \qquad \text{and} \qquad \sigma_k(v) \geq j \qquad (3)$$

The fact that y_i is not copied to port O_2 of cell k and x_j is not copied to port O_1 of cell q implies that $i \geq v$ and $j \geq \sigma_q(\ell)$. But $i > v$ may only be satisfied if the previous input on I_3 of cell k, say $a_{u,\sigma_k(u)}$, satisfies $u = i-1$, which means that $u \geq v$ and contradicts P1. Similarly, we may show that $j > \sigma_q(\ell)$ contradicts P2. Hence

$$i = v \qquad \text{and} \qquad j = \sigma_q(\ell) \qquad (4)$$

From (3) and (4), we get

$$v \leq \ell \quad \text{and} \quad \sigma_k(v) \geq \sigma_q(\ell)$$

which contradicts Definition 2 for $S_k < S_q$, and hence, given that k<q, contradicts property P3 of the input.

3.3. Pseudo systolic synchronization

In order to study the behavior and estimate the execution time of the network MAT/VEC , we follow the technique suggested in [13] for the study of self-timed computations. Namely, we introduce a simpler, hypothetical, computation (called a pseudo systolic computation) that is obtained by forcing some synchronization on the self-timed computation. The additional synchronization may only slow down execution, and hence the execution time of the pseudo systolic computation forms an upper bound on the execution time of the self-timed computation.

A pseudo systolic version of the self-timed computation discussed in this section may be obtained by replacing step 3 in CYCLE 1 by the following

```
3) wait for a synchronization signal SYNC ;
   Ry = Ry + Ra * Rx
```

The purpose of the SYNC signal is the synchronization of all the cells such that the execution of the network alternates between two phases; a communication phase and a processing phase. During the communication phase, the data flows in the network until each cell is either blocked waiting for data (in steps 2.1 or 2.2), or waiting for SYNC (in step 3). We assume that all the cells are connected to a hypothetical controller that issues the signal SYNC after it detects the termination of the communication phase. At that instant, all the cells that are waiting in step 3 perform the multiplication simultaneously, while the other cells remain idle. This is the processing phase. A communication phase followed by a processing phase is called a global cycle of the network. In this paper, we let N be the total number of global cycles needed to terminate the computation, and we denote by CP_t and PP_t, $t=1,\ldots,N$, the computation phase and the processing phase, respectively, of the global cycle t.

Given that external data on I_3, I_4 and I_5 are available when needed, we may define the function $\alpha : [-\pi_1,\pi_2]\times[1,N] \rightarrow A$ such that $\alpha(k,t)$ is the element of A that is stored in the register Ra of cell k (read from port I_3) during the processing phase PP_t. Although, for any specific t, $\alpha(k,t)$ is defined for all k cells, some cells will be idle during PP_t, and hence, will not operate on $\alpha(k,t)$. Let M_t be the set of cells that are not idle during PP_t, and define the t^th computation front CF_t as the set of elements of A that are operated upon during that phase. More specifically,

$$CF_t = \{\alpha(k,t) : k \in M_t\}$$

The succession of computation fronts represents the progress in the execution of the pseudo systolic computation. More specifically, given a certain matrix, we may connect the elements of each computation front by a piecewise linear curve and thus obtain a visual picture that describes the propagation of the computation. For example, we show in Fig 4 the different computation fronts that result from the pseudo systolic execution of MAT/VEC on the matrix A of Fig 1.a. For clarity, we represent each non zero elements, $a_{i,j} \in S_k$, of A by its stripe number k, and we represent the zero elements of A by dots. Note that the concept of computation fronts is the same as that suggested in [17]. *However, by allowing irregular fronts, we are able* to model data driven computations that depend on the value of the input as well as its availability.

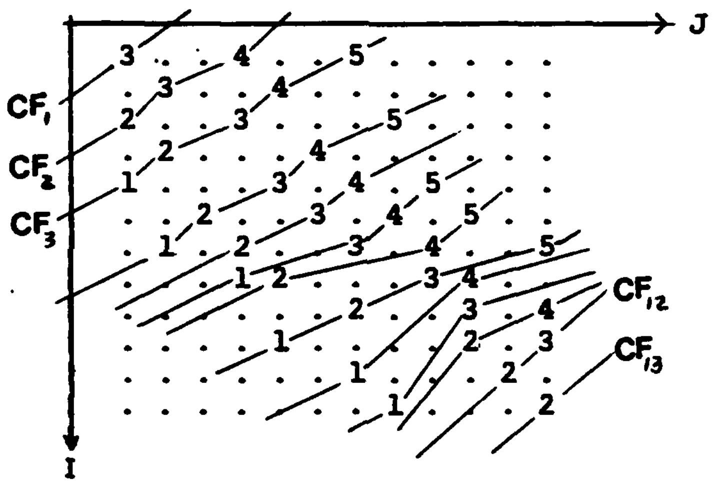


Fig 4 - Computation fronts

Computation fronts may be constructed systematically if the conditions that governs the relation between the elements of the fronts are known. In order to derive these conditions, we first observe that if $a_{i,\sigma_k(i)}$ is in CF_t, then both y_i and $x_{\sigma_k(i)}$ should be at cell k during the processing phase PP_t. Similarly, if $a_{\ell,\sigma_q(\ell)}$ is in the same front CF_t, then, both y_ℓ and $x_{\sigma_q(\ell)}$ should be at cell q during PP_t. Assuming that $q > k$, then the sequential order of the x and y data streams requires that $\ell < i$ and $\sigma_q(\ell) > \sigma_k(i)$, respectively. In other words the following should be satisfied

<u>Consistency of data flow condition</u>: If $a_{\ell,\sigma_q(\ell)}$ and $a_{i,\sigma_k(i)}$, $q > k$, are in the same computation front, then

$$\ell < i \qquad \text{and} \qquad \sigma_q(\ell) > \sigma_k(i) \tag{5.a}$$

If the queues on the communication lines have infinite capacity, then any number of data items may be buffered between cells k and q. That is, there is no upper limit on the values of $(i-\ell)$ or $(\sigma_q(\ell)-\sigma_k(i))$, and hence (5.a) is the only necessary condition for $a_{i,\sigma_k(i)}$ and $a_{\ell,\sigma_q(\ell)}$ to be in the same front. More descriptively, (5.a) means that every line segment in a computation front should have a slope s on the J axis (see Fig 4) that satisfies

$$-\infty < \tan(s) < 0$$

that is

$$90^\circ < s < 180^\circ \tag{5.b}$$

In addition to the condition imposed on each individual front, we have to ensure that the fronts propagate in the same direction. More specifically,

<u>Unidirectional propagation condition</u>: If $a_{i,\sigma_k(i)} \in CF_t$ and $a_{\ell,\sigma_k(\ell)} \in CF_\tau$, $t > \tau$, then

$$i > \ell \qquad (6)$$

Now, given the zero pattern of any matrix A, we may construct the computation fronts of A as follows:

```
ALG1 :  /* Construction of the computation fronts */
1) Start from the upper left corner of A and construct $CF_1$,
such that,
   C1) It includes as many nonzero elements of A as possible
   C2) Condition (5) is satisfied, and
   C3) All elements of A enclosed between $CF_1$ and the two
       axes are zeroes (implied by condition (6)).
2) For t=2,3,..., repeat until every non zero element of A
is in some front
  2.1) Given $CF_{t-1}$, construct $CF_t$ such that
   C1) It includes as many nonzero elements of A as possible
   C2) Condition (5) is satisfied, and
   C3) All elements of A enclosed between $CF_{t-1}$ and $CF_t$
       are zeroes (implied by condition (6)).
```

By the definition of the pseudo systolic network, all possible communications are performed before the beginning of a processing phase. Moreover, every cell that receives all its operands during the communication phase, executes step 3 of CYCLE 1 upon reception of the SYNC signal. For these reasons, we construct the computation fronts by including in each front as many elements of A as possible.

Large matrices that appear in practical applications have usually non zero diagonal elements. For this type of matrices, we may establish the following lower bound:

Proposition 1: If A is an n×n matrix with non zero diagonal elements, then at least n computation fronts are required in order to cover all the nonzero elements of A.

Proof: Each diagonal element should be in some front, and condition (5) does not allow a single front to include more than one diagonal element. []

In order to establish an upper bound on the number of computation fronts, we study the question of not including in each front as many elements of A as possible. More specifically, assume that during the construction of CF_t, a particular element $a_{i,\sigma_k(i)}$ can be included in CF_t. This means that at the end of the communication phase CP_t, cell k is waiting in step 3 of CYCLE 1. The exclusion of $a_{i,\sigma_k(i)}$ from CF_t may only result if cell k remains idle during the processing phase PP_t, say because SYNC did not reach that cell due to some transmission error. Although this error does not cause a failure of the computation, it does slow it down because cell k will stay at step 3 of CYCLE 1 waiting for the SYNC signal of the next global cycle.

Hence, any set of computation fronts that satisfy conditions (5) and (6) corresponds to some execution of the pseudo systolic network with unreliable broadcast of SYNC. In order to reserve the term computation fronts to the sets that are constructed by ALG1 and that correspond to the execution of a reliable pseudo systolic network, we introduce the following definition:

Definition 4: If condition C1 is removed from ALG1, then any set of fronts that result from the construction is called a set of contours of the matrix A.[]

Clearly, the number of computation fronts that cover A is less than or equal to the number of contours in any set of contours that covers A. This may be restated as follows:

Proposition 2: Given a matrix A, If we may include all the non zero elements of A in $\bar{N}$ contours that satisfy (5) and (6), then the pseudo systolic execution of MAT/VEC terminates in at most $\bar{N}$ global cycles.

In the next section we study a type of matrices for which the upper bound on the number of computation fronts provided by Proposition 2 coincides with the lower bound established by Proposition 1.

4. MATRICES WITH NON-OVERLAPPING STRIPES

Let S_1 and S_2 be two full stripes. By Definition 2, $S_1 < S_2$ if $\sigma_1(i)<\sigma_2(i)$, $i=1,\dots,n$. A more restrictive condition may be obtained if we require that, for every $i=2,\dots,n$, the intervals $\sigma_1(i)-\sigma_1(i-1)$ and $\sigma_2(i)-\sigma_2(i-1)$ do not overlap. That is, if

$$\sigma_1(i) \leq \sigma_2(i-1) \qquad i=2,\dots,n$$

The following definition extends this simple condition to stripes that are not full.

<u>Definition 5</u>: The π stripes of a matrix A are said to be non-overlapping, if for any stripe S_k, $-\pi_1 \leq k \leq \pi_2$, and any element $(i,\sigma_k(i))\in S_k$, we have

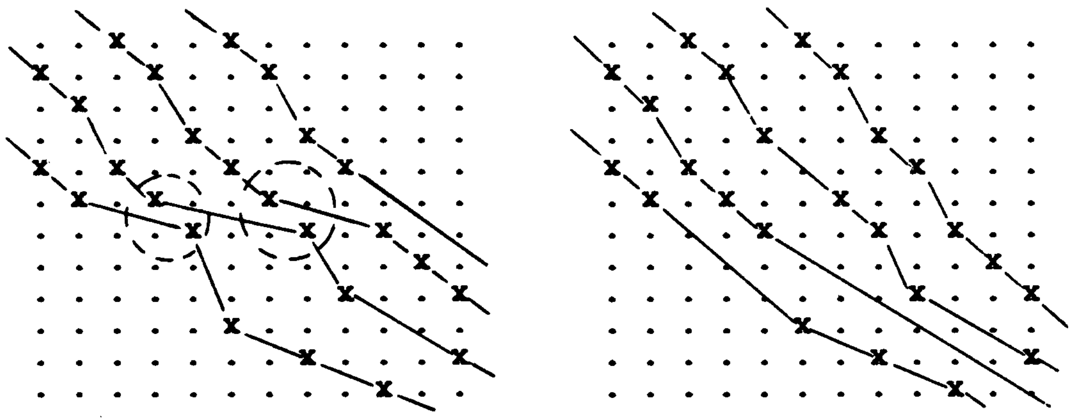
$$\sigma_k(i) \leq \sigma_{k+m}(i-m) \tag{7}$$

where m is the smallest positive integer such that $(i-m,\sigma_{k+m}(i-m)) \in S_{k+m}$. If the inequality in (7) is strict, that is $<$ replaces $\leq$, then the stripes of A are called strictly non overlapping. []

For example, the matrix shown in Fig 5.b has strictly non overlapping stripes, while the matrix shown in Fig 5.a has overlapping stripes. The positions where overlap occurs are indicated on the figure.

The following lemma may be easily proved by induction on (7).

Lemma 1: The π stripes of A are non-overlapping if and only if for any k, $-\pi_1 \le k \le \pi_2$, and integers i and m, such that $(i,\sigma_k(i)) \in S_k$, and $(i-m,\sigma_{k+m}(i-m)) \in S_{k+m}$, equation (7) is satisfied.[]



(a) A matrix with overlapping stripes

(b) The same matrix with strictly non overlapping stripes

Fig 5

The property of strictly non overlapping stripes guarantees that if both $a_{i,\sigma_k(i)}$ and y_i are at cell k during a specific global cycle t, then $x_{\sigma_k(i)}$ may not be locked behind another cell k+m, m>0, and hence should arrive at cell k during the same global cycle t. In other words, the computation is not delayed due to internal data conflicts. This is formalized by the following proposition:

Proposition 3: Let A be a matrix with non-zero diagonal elements. If A is striped such that all its diagonal elements are covered by one stripe and all its stripes are strictly non-overlapping, then the pseudo systolic computation of MAT/VEC, with input matrix A, terminates in exactly n global cycles.

Proof: Let $\pi=\pi_1+\pi_2+1$ be the stripe count of A, and denote the stripes of A by S_k, $k=-\pi_1,\dots,0,\dots,\pi_2$, where $S_k<S_{k+1}$ and S_0 is the stripe that contains all the diagonals. Construct for each $r=1,\dots,n$ a contour C_r that includes $a_{r,r}$ and has a slope of one stripe per row. That is C_r includes for each $k=-\pi_1,\dots,\pi_2$, the element (if any) of stripe k which is at row r-k. More specifically,

$$C_r = \{a_{r-k,\sigma_k(r-k)} : -\pi_1<k<\pi_2 \text{ and } (r-k,\sigma_k(r-k)) \in S_k\} \quad (8)$$

For any specific $a_{i,j}\neq 0$, there exists a unique k such that $a_{i,j} \in S_k$, that is $\sigma_k(i)=j$. Hence, there exists exactly one contour that includes $a_{i,j}$, namely C_{i+k}. In other words, the contours C_r, $r=1,\dots,n$ cover all the non zero elements of A. Moreover, If $a_{v-k,\sigma_k(v-k)} \in C_v$ and $a_{u-k,\sigma_k(u-k)} \in C_u$ are in the same stripe S_k, then $v-k>u-k$ implies that $v>u$. That is the condition of unidirectional propagation, namely equ (6), is satisfied.

It remains to prove that the consistency of data flow condition, namely equ. (5), is satisfied. Let $(i,\sigma_k(i))$ and $(\ell,\sigma_q(\ell)$ be any two elements in C_r. If $q > k$, then from the definition of C_r, $i=r-k$ and $\ell=r-q$, and hence $\ell < i$. But the stripes of A are strictly non overlapping, and thus, by using $m=q-k$ in (7), we obtain

$$\sigma_k(i) < \sigma_{k+q-k}(i-(q-k)) = \sigma_q(r-k-q+k) = \sigma_q(\ell)$$

which satisfies (5), and completes the proof that all the non zero elements of A may be covered by n contours that satisfy (5) and (6). The result of the proposition, then, follows directly from Propositions 1 and 2. []

Proposition 3 applies only to matrices with strictly non overlapping stripes. A similar result may be obtained for non overlapping stripes, if we weaken the condition on computation fronts (contours) such that the line segments of a specific front (contour) may be vertical. That is $\sigma_q(\ell)$ and $\sigma_k(i)$ in (5.a) may be equal, which means that a component of the x data stream may be at two different cells k and q during the same processing phase. This may be achieved if each cell in MAT/VEC writes, immediately, on O_1 the value of x that it reads from I_1. More precisely, the operation of each cell (CYCLE 1) should be modified to the following:

```
CYCLE 2:        /* Initially, CX=CY=0 */
    1) Ra = [$I_3$] ; Rj = [$I_4$] ; Ri = [$I_5$]
    2) Do steps 2.1 and 2.2 in parallel
       2.1) wait until data is available on $I_1$
            Rx = [$I_1$] ; $O_1$ ← Rx ; CX = CX + 1
            If CX < Rj Then Go To 2.1
                       Else JOIN step 2.2.
       2.2) wait until data is available on $I_2$
            Ry = [$I_2$] ; CY = CY + 1
            If CY < Ri Then  { $O_2$ ← Ry ; Go To 2.2 }
```

```
        Else JOIN step 2.1.
3) Ry = Ry + Ra * Rx
4) $O_2 \leftarrow$ Ry.
```

The proof of the following proposition is very similar to that of Proposition 3.

Proposition 4: Let A be a matrix with non-zero diagonal elements, that is striped such that all its diagonal elements are covered by one stripe and all its stripes are non-overlapping. If each cell executes CYCLE 2, then the pseudo systolic computation of MAT/VEC, with input A, terminates in n global cycles.[]

Given a sparse matrix A, the advantage of constructing a non-overlapping stripe structure for A is clear. However, assuming that π is minimum number of stripes that may cover A, and π_n is the number of non-overlapping stripes that may cover A, then, usually, $\pi_n > \pi$. Hence, a trade off should be considered between 1) using a network with π_m cells that terminates execution in n global cycles, or 2) using a network with π cells which requires an execution time larger than n global cycles.

The cost of the determination of a non-overlapping stripe structure for general sparse matrices is usually high. More specifically, the modification of the algorithm given in the appendix such that to include condition (7) requires some form of back-tracking, which is costly. However, for some type of matrices, non-overlapping stripes may be obtained for very low additional costs. For example, for the class of finite element stiffness matrices considered in [12], non-overlapping stripes may be obtained by renumbering the nodes of the grid from which the matrix is generated.

Besides the number of global cycles, the execution time of MAT/VEC, is determined by the time for the completion of each global cycle, which depends on the communication activities that takes place during the communication phase of the cycle. We consider these activities in some details.

5. The communication issue

By the definition of pseudo systolic networks, no communication takes place during the processing phases of global cycles. Hence, during a specific processing phase PP_t, data assumes a static profile. In other words, a function may be defined for each data stream to specify the data items at each computational cell.

For example, consider the x data stream in MAT/VEC and assume that the register Rx is set initially to an arbitrary value, say x_0 (the value of x_0 is irrelevant to the computation). The x-stream profile at the processing phase PP_t may then be specified by a function $xP_t:[-\pi_1,\pi_2]\rightarrow[0,n]$ such that $xP_t(k) = j$, where x_j is the value of the register Rx in cell k during PP_t. The y-stream profile at PP_t may be specified by a similar function $yP_t:[-\pi_1,\pi_2]\rightarrow[0,n]$. Note that the registers Rx and Ry contain always some values, and hence xP_t and yP_t are defined for every cell k.

If $k\epsilon M_t$, that is cell k is not idle during PP_t, then $xP_t(k)$ and $yP_t(k)$ may be determined from the computation front CF_t. More specifically, if $k\epsilon M_t$, then $a_{\ell,\sigma_k(\ell)} \in CF_t$ for some ℓ, and hence, y_ℓ and $x_{\sigma_k(\ell)}$ are at cell k during PP_t. That is

$$a_{\ell,\sigma_k(\ell)} \in CF_t \Rightarrow yP_t(k)=\ell \text{ and } xP_t(k)=\sigma_k(\ell) \qquad (9)$$

We call the values of $xP_t(k)$ and $yP_t(k)$, for $k\epsilon M_t$, the knots of the profiles. On the other hand, if $k\notin M_t$, then the values of $xP_t(k)$ and $yP_t(k)$ may not be determined from a simple formula. However, from the specification of the operation of each cell (CYCLE 1), it is clear that the following properties should be satisfied for $k=-\pi_1,\ldots,\pi_2-1$ and any t:

$$xP_t(k) \begin{cases} < xP_t(k+1) & \text{if } k+1 \in M_t \\ \leq xP_t(k+1) & \text{if } k+1 \notin M_t \end{cases} \tag{10.a}$$

$$yP_t(k) \begin{cases} > yP_t(k+1) & \text{if } k \in M_t \\ \geq yP_t(k+1) & \text{if } k \notin M_t \end{cases} \tag{10.b}$$

Note that it is possible that $xP_t(k)=xP_t(k+1)$ if $k+1 \notin M_t$. More specifically, when cell k+1 is waiting for a new input, its register Rx keeps the old value of x that has been written on O_1 during CP_t. If this value is also read by cell k during CP_t, then the registers Rx in both cells k and k+1 contain the same value during PP_t.

Also, the elements of each stream arrive at a specific cell in order. That is, the following is satisfied for $k=-\pi_1,\ldots,\pi_2$ and any t:

$$xP_t(k) \begin{cases} < xP_{t+1}(k) & \text{if } k \in M_t \\ \leq xP_{t+1}(k) & \text{if } k \notin M_t \end{cases} \tag{11.a}$$

$$yP_t(k) \begin{cases} < yP_{t+1}(k) & \text{if } k \in M_t \\ \leq yP_{t+1}(k) & \text{if } k \notin M_t \end{cases} \tag{11.b}$$

Again, the equalities may hold because a register does retain its value if it is not overwritten by a new one.

Equations (10) and (11) force on data profiles the same conditions that equations (5) and (6) force on computation fronts. In fact, it is straight forward to derive (5) and (6) from (10) and (11). Moreover, given some computation fronts which satisfy (5) and (6), that is which simulate an execution of MAT/VEC, there should exist some functions that satisfy (9), (10) and (11) and correspond to the data profiles during execution. However, the mathematical construction of these functions is complex and involves the solution of a set of simultaneous inequalities, namely (10) and (11). For this reason, we seek further restrictions on the computation fronts and data profiles, by limiting the communication capabilities of the network.

5.1. Communication links with limited buffer capacity

A communication link directed from a cell k to cell k+1 may be regarded as a queue. Only cell k may append to this queue and only cell k+1 may access (and delete) its front element. So far, we have assumed that the communication queues (buffers) in MAT/VEC have infinite capacity, that is $d_x = d_y = \infty$, where d_x and d_y are the capacities of individual queues on the x-stream and y-stream, respectively. With this assumption, we were able to derive the conditions (5) and (6) which enable us to construct the computation front for any given matrix. Clearly, any limitation on d_x or d_y represents some additional restrictions that should be taken into account during that construction.

More specifically, and without going into the implementation details of the communication protocols, if x_u and x_v are at cells k and k+1, respectively, during the processing phase PP_t, then v-u elements of the x-stream should be buffered between the two cells, which requires a queue capacity of, at least, that size. That is

$$xP_t(k+1) - xP_t(k) \leq d_x \tag{12.a}$$

Similarly, for the y-stream

$$yP_t(k) - yP_t(k+1) \leq d_y \tag{12.b}$$

Equations (12) may be translated into restrictions on computation fronts. More precisely, we may derive the following from (9) and (12):

<u>Buffer capacity condition</u>: If $a_{\ell,\sigma_q(\ell)}$ and $a_{i,\sigma_k(i)}$, $q > k$, are in the same computation front CF_t, then

$$\sigma_q(\ell) - \sigma_k(i) \leq (q-k)\ d_x \tag{13.a}$$

and

$$i - \ell \leq (q-k)\ d_y \tag{13.b}$$

The buffer capacity condition is weaker than conditions (12) because it restricts the collective capacity of the links between cells q and k, rather that the capacities of the individual links. In order to clarify this point let $d_x=3$, $q=k+2$, $i=\ell+2$, and $\sigma_{k+2}(i-2)-\sigma_k(i)=6$. Clearly, with these values, (13.a) is satisfied, and by definition, $xP_t(k+2)=\sigma_{k+2}(i-2)$ and $xP_t(k)=\sigma_k(i)$. Although both (12.a) and (13.a) specify that at most $2d_x=6$ data elements may be buffered between cells k and k+2, only (12.a) specifies that three of these elements should be buffered between cells k and k+1 and the other three between cells k+1 and k+2. More specifically, (13.a) does not put any restriction on $xP_t(k+1)$, while (12.a) requires that $xP_t(k+1)=\sigma_k(i)+3$. Now, let $a_{i-1,\sigma_{k+1}(i-1)}$ be in the next computation front CF_{t+1} with $\sigma_{k+1}(i-1)=\sigma_k(i)+2$, that is $xP_{t+1}(k+1)=\sigma_k(i)+2$. It is easy to see that the above data is inconsistent because $xP_t(k+1) > xP_{t+1}(k+1)$, which violates (11.a). However, by allowing arbitrary values to $xP_t(k+1)$, the buffer capacity condition (13.a) does not detect this inconsistency.

If conditions (13) are added to ALG1 of Section 3, then computation fronts that satisfy (5), (6) and (13) may be constructed for any given matrix. However, because (13) is weaker than (12), the constructed fronts represent the execution of MAT/VEC only if it is possible to find a corresponding data profile (with knots specified by (9)) that satisfy (10), (11) and (12). Although this technique of constructing the computation fronts and then checking that they represent the actual execution of the network may, in general, fail, it can be used to show that the results of Propositions 2 and 3 are independent of the size of the queues on the y-stream links.

More specifically, consider the minimum value of d_y, namely $d_y=1$, and keep $d_x=\infty$. It is easy to check that the contours C_r, $r=1,\ldots,n$, used in the proof of Proposition 3 do satisfy (13.b) with $d_y=1$. Moreover, let

$$yP_r(k) = r-k \qquad r=1,\ldots,n, \quad k=-\pi_1,\ldots,\pi_2 \tag{14}$$

The knots of this profile corresponds to C_r (as specified by (9)). Also, (14) satisfies (10.b), (11.b) and (12.b) with $d_y=1$. Hence, the n contours C_r given by (8) correspond to some execution of MAT/VEC with $d_y=1$.

In other words, if $d_y=1$ and $d_x=\infty$ in MAT/VEC, then the execution of the network for any n×n matrix with non zero diagonal elements does terminate in n global cycles. Although this is a good result, it is preferable to replace the condition $d_x=\infty$ by one of the form $d_x \geq d_{min}$. In order to derive the lower bound d_{min}, we should construct an x-stream profile that satisfies (9), (10.a) and (11.a), and then from (12.a) get

$$d_{min}= \max_{r,k}\{xP_r(k+1)-xP_r(k) \mid r=1,..,n, \; k=-\pi_1+1,..,\pi_2\} \tag{15}$$

For general striped matrices, the construction of such x-stream profile seems difficult. However, for certain types of stripe matrices the construction is straight forward (see [12] for examples).

In order to give further meaning to the bound (15), it is useful to consider matrices with full, non-overlapping, stripes. In this case, it is easy to see that the contours given by (8) are the actual computation fronts, that is

$$CF_t = \{a_{t-k,\sigma_k(t-k)} \mid k=-\pi_1,\ldots,\pi_2 \} \tag{16}$$

From (9), the corresponding x-stream profile is given by

$$xP_t(k) = \sigma_k(t-k) \tag{17}$$

which by the very properties of the stripes satisfy (10.a) and (11.b). Now, from (15),

$$\begin{aligned} d_{min} &= \max_{k,t}\{\sigma_{k+1}(t-k-1) - \sigma_k(t-k)\} \\ &< \max_{k,t}\{\sigma_{k+1}(t-k) - \sigma_k(t-k)\} \\ &= \text{the maximum separation between the} \\ &\quad\ \text{stripes of the matrix.} \end{aligned}$$

In other words, the separation between the stripes determines the minimum size of the queues needed on the x-stream communication links.

Finally, we note that, with finite queues capacity, the communication protocol should not allow a cell to write on a queue that is full. More specifically, with $d_y=1$, the statement $O_2 \leftarrow Ry$ in CYCLE 1, should be interpreted as "wait until the queue is not full, then write the content of Ry".

5.2. Communication time in MAT/VEC

Let τ_m be the time required by a cell in MAT/VEC to complete a floating point operation (step 3 of CYCLE 1), and let τ_c be the time required to move a data item from the input port of some cell to that of the next cell. For example, a data item, say x_j, may be moved from port I_1 of cell k to port I_1 of cell k-1 in time τ_c. This includes the time for the execution of step 2.1 of CYCLE 1 and the associated protocols, as well as the time required for the signals to travel on the communication lines.

In order to estimate the execution time of any specific global cycle, we assume that cell k executes steps 2.1 $\xi_t(k)$ times and step 2.2 $\eta_t(k)$ times during the communication phase CP_t of the tth global cycle. Clearly, $\xi_t(k)$ and $\eta_t(k)$ may be estimated from the data profiles as follows:

$$\xi_t(k) = xP_t(k) - xP_{t-1}(k) \qquad (18.a)$$
$$\eta_t(k) = yP_t(k) - yP_{t-1}(k) \qquad (18.b)$$

However, each cell in MAT/VEC has to wait until all the other cells complete their communication activities. Hence, the duration of the communication phase CP_t is determined by the maximum of $\xi_t(k)$ and $\eta_t(k)$ for all k. That is by

$$\xi_t = \max\,\{\xi_t(k) \mid k=-\pi_1,\ldots,\pi_2\} \qquad (19.a)$$

and

$$\eta_t = \max\,\{\eta_t(k) \mid k=-\pi_1,\ldots,\pi_2\} \qquad (19.b)$$

From CYCLE 1, it is now easy to see that the time required for the completion of global cycle t is

$$T_t = \tau_m + \max\{\xi_t\, ,\, \eta_t\}\, \tau_c$$

For d_y=1 and $d_x \geq d_{min}$, the y-stream profile is given by (14), from which we find that η_t=1, and hence the total execution time of MAT/VEC is

$$T = \sum_{t=1}^{N} T_t = N\,\tau_m + \tau_c \sum_{t=1}^{N} \xi_t \qquad (20)$$

The values of ξ_t, t=1,...,N, depend on the specific stripe structure of the input matrix. However, without knowing the specific stripe structure of the matrix, we may only bound the execution time of MAT/VEC by

$$T < N\,(\tau_m + \xi_{max}\,\tau_c)$$

where

$$\xi_{max} = \max\{\xi_t \mid t=1,\ldots,N\}$$

As we did in the last section, we may give further meaning to ξ_{max} by considering matrices with full, non-overlapping, stripes. For this type of matrices, the x-stream profile is given by (17), from which we obtain

$$\xi_{max} = \max_{k,t}\{\sigma_k(t-k)-\sigma_k(t-k-1)\} = \max_{k,i}\{\sigma_k(i)-\sigma_k(i-1)\}$$

= *the maximum slope of any stripe in the matrix.*

Note that equation (20) estimates the execution time of MAT/VEC assuming the hypothetical pseudo systolic synchronization. In actual execution, however, the synchronization of MAT/VEC should be merely data driven (no wait in step 3 of CYCLE 1), and hence faster than the pseudo systolic execution. In other words, the value of T given by (20) is an upper bound for the actual execution time of MAT/VEC. This upper bound is used in [12] to study the performance of MAT/VEC for finite element matrices.

6. The iterative solution of sparse linear systems

In this section, we consider one of the most efficient iterative techniques for the solution of linear systems of the form $Ax=z$ Namely the preconditioned conjugate gradient method. This method applies conjugate gradient iterations to the system $M^{-1}A\,x = M^{-1}z$, where the preconditioner matrix M is a suitable approximation of A^{-1}. In many preconditioning techniques, such as incomplete factorizations [11] and SSOR [1], the matrix M may be expressed as the product of a unit lower triangular matrix L, a diagonal matrix D, and a unit upper triangular matrix U. That is $M = LDU$. The property that makes these preconditioners attractive is that the matrices L and U have the same zero structures as the lower and upper parts of the matrix A, respectively.

The bulk of the work in each iteration of the preconditioned conjugate gradient method is the multiplication of the matrix A by a vector, and the solution of two triangular linear systems of the forms $Ly=u$ and $Uz=v$.

It was shown in the previous sections that, if a suitable stripe structure is found for the matrix A, then the multiplication of A by any vector may be efficiently executed in parallel on the linear network MAT/VEC. Here, we show that, with very simple modification, MAT/VEC may also be used for the solution of any unit lower triangular system $Ly=u$. The unit upper triangular system $Uz=v$, may also be viewed as a unit lower triangular system $U^Tz'=v'$, where z'

and v' are obtained by reversing the elements of z and v, respectively. That is, if the order of U is n, then $z'_i=z_{n-i+1}$ and $z'_i=v_{n-i+1}$.

Assume, as before, that A has $\pi_1+\pi_2+1$ stripes $S_{-\pi_1}<\ldots<S_{\pi_2}$, where S_0 is a full stripe that contains all the diagonal elements. Hence, L has π_1+1 stripes that coincide with those in the lower triangular part of A. Namely, $S_{-\pi_1},\ldots,S_0$.

Let MAT/VEC, with $d_y=1$ and $d_x\geq d_{min}$, be applied to the multiplication of L by any vector (Fig 6). In this case, the inputs on ports I_4 and I_5 of cell 0 are not needed because all the elements in the full stripe $S_0=\{(i,i)|i=1,\ldots,n\}$ are supplied, in order, to port I_3. For the same reason, the counters CX and CY of CYCLE 1 are not needed in cell 0. In other words, the cycle of cell 0 may be described by

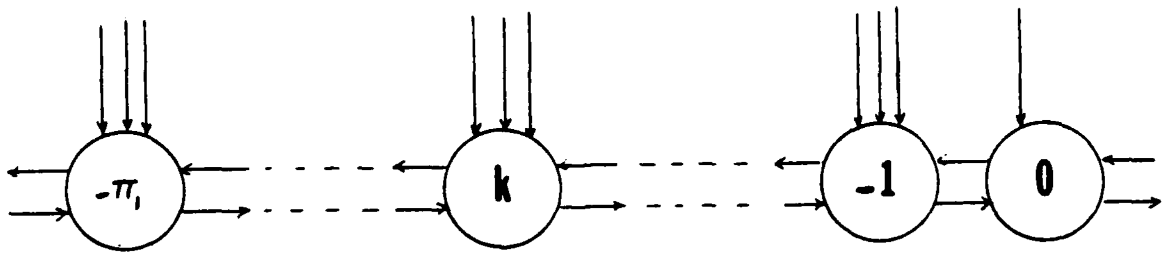


Figure 6 - A modified version of MAT/VEC

<u>CYCLE 3:</u>

1) Ra = $[I_3]$

```
2) wait until data is available on $I_1$ and $I_2$ ;
   $Rx = [I_1]$ ; $Ry = [I_2]$
3) $Ry = Ry + Ra * Rx$
4) $O_1 \leftarrow Rx$ ; $O_2 \leftarrow Ry$.
```

The same network of Fig 6 may be used for the solution of the triangular system $Ly=u$ if the elements of the vector u, instead of the elements of S_0, are supplied to I_3 of cell 0, and the operation of this particular cell is described by the following cycle (instead of CYCLE 3).

<u>CYCLE 4:</u>

```
1) $Ra = [I_3]$
2) wait until data is available on $I_2$
       $Ry = [I_2]$ ;
3) $Ry = -Ry / Ra$
4) $O_1 \leftarrow Ry$ ; $O_2 \leftarrow Ry$.
```

We call the resulting network TRIANG. Note that the input port I_1 of cell 0 in TRIANG is not used, and hence no data need to be supplied on it. Also, the elements of the result vector y are produced on the output port O_2.

The computation fronts for the pseudo systolic execution of TRIANG may be obtained by applying our earlier rules to a matrix L^* which is obtained by replacing the diagonal elements in L by the elements of the vector u. But the zero structure of L^* is identical to that of L, which, in turns, is identical to the lower triangle of A. Hence, the stripe structure of A may be used to estimate the execution time of TRIANG. More specifically, if the stripes of A are strictly non-overlapping, then the pseudo systolic execution of TRIANG terminates in n global cycles. The capacities of the buffers, as well as the communication time, in TRIANG may be also estimated from the stripe structure of A.

Finally, we note that the networks MAT/VEC and TRIANG may be used as two high speed peripheral devices for a host computer that executes the preconditioned conjugate gradient iterations. In fact, only MAT/VEC is needed if a mechanism is available for switching the function of cell 0 between CYCLE 3 and CYCLE 4.

7. CONCLUSION

We introduced the concept of striping a sparse matrix, which is, namely, the inclusion of the non-zero elements of the matrix in a structure which is regular enough to allow for efficient parallel manipulation. Although the concept is general, we only discussed its application to the design of regular VLSI networks for sparse matrices.

The operation of each cell in the networks presented in this paper are data dependent, as well as data-driven. This makes the application of formal analysis models (e.g. [5,14]) extremely difficult. For this reason, the additional simplification of pseudo systolic synchronization was assumed, which allowed for the establishment of upper bounds on the performance of the networks. The actual performance of the data-driven networks may only be estimated by a detailed simulation of the computations.

It is proved that for an $n \times n$ matrix with π non-overlapping stripes, ($\pi << n$) and minimum separation d_{min} between stripes, the multiplication of the matrix by a vector may be performed in n global cycles, using a linear network of π cells connected by links that can buffer d_{min} data items. The same result also applies to the solution of triangular linear systems.

The task of finding a stripe structure for a sparse matrix may be accomplished in many different ways, including the algorithm given in the appendix. However, for some types of matrices, a stripe structure may be naturally determined from the underlying problems. For example, stiffness matrices arising in finite element analysis are usually generated from finite grids, and the stripe structure of a stiffness matrix is directly specified by the neighboring relation between the nodes of the corresponding grid. For more details we refer to [12].

Acknowledgement

I would like to thank Werner Rheinboldt for his many comments that helped in the formalization of the concept of striped matrices.

APPENDIX

An algorithm (in the form of a Fortran subroutine) is presented for the determination of a stripe structure for any given matrix A. The structure is specified by a matrix PA in which each column k specifies the positions of the elements in stripe S_k (see equation 1.b). More precisely, if PA(i,k)=j, then $a_{i,j} \in S_k$, and if PA(i,k)=0, then, there is no elements in row i that belong to S_k. Note that, for simplicity, we replaced $-b_1$ in (1.b) by 0.

The linear array "Last" should be set, in the calling program, such that Last(i), $1 \leq i \leq n$, contains the number of non-zero elements in row i of the matrix A. Also, the integer "nstrip" should be set to the maximum of Last(i), i=1,...,n. The column numbers of the "Last(i)" elements in row i of A are initially stored in the first "Last(i)" locations of row i of PA (see Fig 7). The subroutine, then, transforms this input form of PA into the one that specifies a stripe structure of A. The number of stripes is also returned in the variable "nstrip". Following is the algorithm:

```
      subroutine stripes(PA,last,n,nstrip)
      integer PA(n,1), last(n)
c
      j = 0
 10   j = j + 1
        do 1000 i=2,n
c
c         ** Find the previous element in the current stripe    **
 200      m = 0
```

```
x x . . x . .          1 2 5 0          0 1 2 5
. x . x . x .          2 4 6 0          0 2 4 6
x . x . . . .          1 3 0 0          1 3 0 0
. . x x x . x          3 4 5 7          3 4 5 7
. . . . x . x          5 7 0 0          0 5 7 0
. . . x . x .          4 6 0 0          4 6 0 0
. . . . . x x          6 7 0 0          6 7 0 0
```

(a) the matrix A   (b) PA at input   (c) PA at output

Fig 7 - Inputs and outputs of subroutine stripes

```
 300      m = m + 1
          if(i-m .LT. 0) go to 1000
          if(PA(i-m,j) .EQ. 0) go to 300
c         ** If stripe is not strictly increasing, then shift  **
c         ** row i-m by one position starting at column j      **
          if(PA(i-m,j) .GE. P(i,j)) then
            call shift(PA,last,n,nstrip,i-m,j)
            go to 300
          endif
c
 1000   continue
c
      if(j .LT. nstrip) go to 10
c
      return
      end
c*************************************************************
      subroutine shift(P,last,nstrip,n,i,j)
      integer P(n,1),last(n)
c
      last(i) = last(i) + 1
      jt = last(i)
      if(jt .LT. nstrip) nstrip = nstrip + 1
 10     P(i,jt) = P(i,jt-1)
        jt = jt - 1
        if(jt .GT. j) go to 10
      P(i,j) = 0
      return
      end
c*************************************************************
```

Finally, we note that it is possible to modify the above algorithm such that the resulting stripes are non-overlapping. However, it seems impossible to enforce the condition of strictly non-overlapping stripes. In fact, the existence of a stripe structure with strictly non-overlapping stripes is not always guaranteed. The matrix of Fig 7(a) is an example for which such structure does not exist.

REFERENCES

[1] L. Adams, "Iterative Algorithms for Large Sparse Linear Systems on Parallel Computers," Ph.D. Thesis, Univ. of Virginia (October 1982).

[2] L. Adams and R. Voigt, "Design, Development and Use of the Finite Element Machine," ICASE Report-172250, NASA-Langley Research Center (October 1983).

[3] H. Amano, T. Boku, T. Kudoh and H. Aiso, "A New Version of the Sparse Matrix Solving Machine," Proc. of the 12th International Symp. on Computer Architecture (June 1985), pp. 100-107.

[4] C. Arnold, M. Parr and M. Dewe, "An Efficient Parallel Algorithm for the Solution of Large Sparse Linear Matrix Equations," IEEE Trans. on Computers, C-32 (March 1983), pp. 265-272.

[5] M. Chen, "Space-Time Algorithms: Semantics and Methodology," Ph.D. Thesis, California Institute of Technology (May 1983).

[6] P. Concus, G. Golub and D. O'Leary, "A Generalized Conjugate Gradient Method for the Numerical Solution of Elliptic PDE's," in Sparse Matrix Computations. Ed. by J. Bunch and D. Rose, Academic Press (1976).

[7] D. Evans, "On Preconditioned Iterative Methods for Partial Differential Equations," in Preconditioning Methods, Theory and Applications. Ed. by D. Evans, Gordon & Breach Science Publishers (1982).

[8] A. George and J. Liu, "Computer Solutions of Large Sparse Positive Definite Systems," Prentice-Hall series in Computational Mathematics (1981).

[9] H. T. Kung and C. E. Leiserson, "Systolic Arrays for VLSI," in Introduction to VLSI Systems (1980). Ed. by C. Mead and L. Conway, Addison-Wesley, Reading, Mass.

[10] S. Y. Kung, K. S. Arun, R. J. Gal-Ezer and D. B. Rao, "Wavefront Array Processor: Language, Architecture and

Applications," IEEE Trans. on Computers, C-31 (November 1982), pp. 1056-1066.

[11] T. Manteuffel, "An Incomplete Factorization Technique for Positive Definite Linear Systems," Mathematics of Computation 34-150 (April 1980), pp. 673-697.

[12] R. Melhem, "Parallel Solution of Linear Systems with Striped Sparse Matrices; Part 2: Stiffness Matrices; A Case Study," Tech. Report. ICMA-86-92 (January 1986).

[13] R. Melhem, "A Study of Data Interlock in VLSI Computational Networks for Sparse Matrix Multiplication," Tech. Report TR-CS-505, Dept. of Comptuer Science, Purdue University (1985).

[14] R. Melhem and W. Rheinboldt, "A Mathematical Model for the Verification of Systolic Networks," SIAM J. on Computing, Vol. 13-3 (August 1984), pp. 341-365.

[15] D. Reed and M. Patrick, "Iterative Solution of Large, Sparse Linear Systems on a Static Data Flow Architecture: Performance Studies," IEEE Trans. on Computers, C-36 (October 1985), pp. 874-880.

[16] D. Reed and M. Patrick, "Parallel, Iterative Solution of Sparse Linear Systems: Models and Architectures," Parallel Computing 2 (1985), pp. 45-67.

[17] V. Weiser and A. Davis, "A Wavefront Notation Tool for VLSI Array Design," in VLSI Systems and Computations (1981). Ed. by H. T. Kung, B. Sproull and G. Steele, Computer Science Press.

[18] O. Wing, "A Content Addressable Systolic Array for Sparse Matrix Computation," J. of Parallel and Distributed Computing 2 (1985), pp. 170-181.